Manual de IMGUI
por
Haylem Candelario Bauzá. Habana Cuba 2025.



Dear ImGui

"Give someone state and they'll have a bug one day, but teach them how to represent state in two separate locations that have to be kept in sync and they'll have bugs for a lifetime." -ryg

(This library is available under a free and permissive license, but needs financial support to sustain its continued improvements. In addition to maintenance and stability there are many desirable features yet to be added. If your company is using Dear ImGui, please consider reaching out.)

Businesses: support continued development and maintenance via invoiced sponsoring/support contracts:
E-mail: contact @ dearimgui dot com
Individuals: support continued development and maintenance here. Also see Funding page.

The Pitch

Dear ImGui is a **bloat-free graphical user interface library for C++**. It outputs optimized vertex buffers that you can render anytime in your 3D-pipeline-enabled application. It is fast, portable, renderer agnostic, and self-contained (no external dependencies).

Dear ImGui is designed to **enable fast iterations** and to **empower programmers** to create **content creation tools and visualization / debug tools** (as opposed to UI for the average end-user). It favors simplicity and productivity toward this goal and lacks certain features commonly found in more high-level libraries. Among other things, full internationalization (right-to-left text, bidirectional text, text shaping etc.) and accessibility features are not supported.

Dear ImGui is particularly suited to integration in game engines (for tooling), real-time 3D applications, fullscreen applications, embedded applications, or any applications on console platforms where operating system features are non-standard.

- Minimize state synchronization.
- Minimize UI-related state storage on user side.
- Minimize setup and maintenance.
- Easy to use to create dynamic UI which are the reflection of a dynamic data set.
- Easy to use to create code-driven and data-driven tools.
- Easy to use to create ad hoc short-lived tools and long-lived, more elaborate tools.
- Easy to hack and improve.
- Portable, minimize dependencies, run on target (consoles, phones, etc.).
- Efficient runtime and memory consumption.
- Battle-tested, used by many major actors in the game industry.

Usage

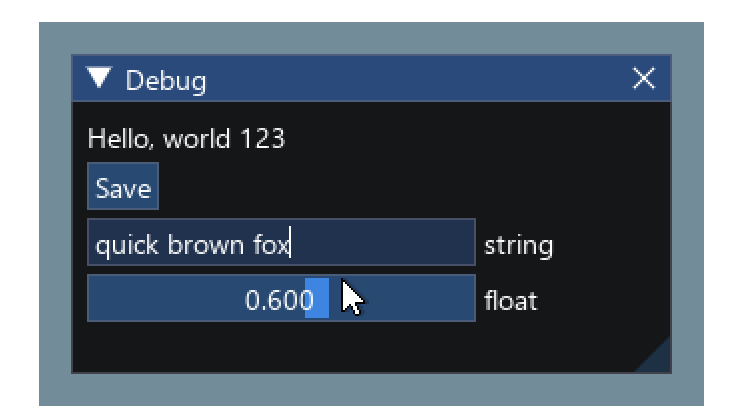
The core of Dear ImGui is self-contained within a few platform-agnostic files which you can easily compile in your application/engine. They are all the files in the root folder of the repository (imgui*.cpp, imgui*.h). **No specific build process is required**. You can add the .cpp files into your existing project.

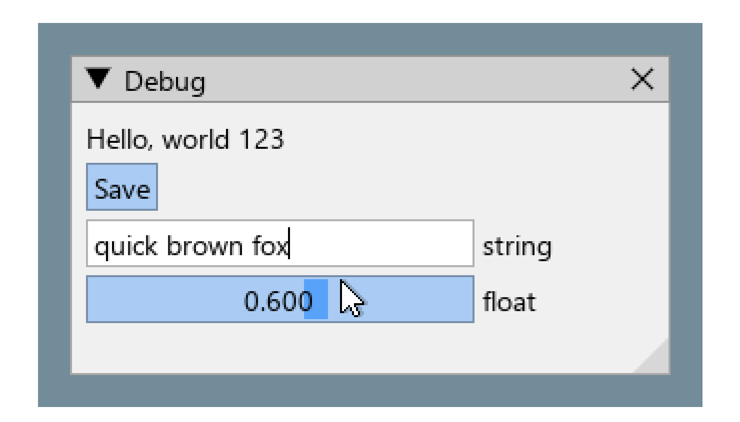
Backends for a variety of graphics API and rendering platforms are provided in the backends/ folder, along with example applications in the examples/ folder. You may also create your own backend. Anywhere where you can render textured triangles, you can render Dear ImGui.

See the Getting Started & Integration section of this document for more details.

After Dear ImGui is set up in your application, you can use it from _anywhere_ in your program loop:

```
ImGui::Text("Hello, world %d", 123);
if (ImGui::Button("Save"))
    MySaveFunction();
ImGui::InputText("string", buf, IM_ARRAYSIZE(buf));
ImGui::SliderFloat("float", &f, 0.0f, 1.0f);
```



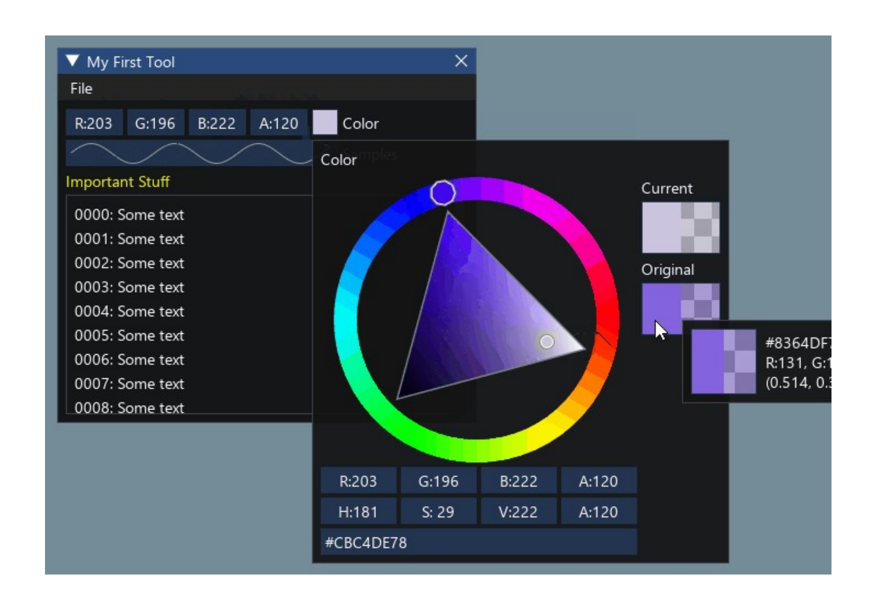



```
// Create a window called "My First Tool", with a menu bar.
ImGui::Begin("My First Tool", &my_tool_active, ImGuiWindowFlags_MenuBar);
if (ImGui::BeginMenuBar())
{
    if (ImGui::BeginMenu("File"))
    {
        if (ImGui::MenuItem("Open..", "Ctrl+O")) { /* Do stuff */ }
        if (ImGui::MenuItem("Save", "Ctrl+S"))   { /* Do stuff */ }
        if (ImGui::MenuItem("Close", "Ctrl+W"))  { my_tool_active = false; }
        ImGui::EndMenu();
    }
    ImGui::EndMenuBar();
}

// Edit a color stored as 4 floats
ImGui::ColorEdit4("Color", my_color);
```

```
// Generate samples and plot them
float samples[100];
for (int n = 0; n < 100; n++)
    samples[n] = sinf(n * 0.2f + ImGui::GetTime() * 1.5f);
ImGui::PlotLines("Samples", samples, 100);

// Display contents in a scrolling region
ImGui::TextColored(ImVec4(1,1,0,1), "Important Stuff");
ImGui::BeginChild("Scrolling");
for (int n = 0; n < 50; n++)
    ImGui::Text("%04d: Some text", n);
ImGui::EndChild();
ImGui::End();
```



Dear ImGui allows you to **create elaborate tools** as well as very short-lived ones. On the extreme side of short-livedness: using the Edit&Continue (hot code reload) feature of modern compilers you can add a few widgets to tweak variables while your application is running, and remove the code a minute later! Dear ImGui is not just for tweaking values. You can use it to trace a running algorithm by just emitting text commands. You can use it along with your own reflection data to browse your dataset live. You can use it to expose the internals of a subsystem in your engine, to create a logger, an inspection tool, a profiler, a debugger, an entire game-making editor/framework, etc.

How it works

The IMGUI paradigm through its API tries to minimize superfluous state duplication, state synchronization, and state retention from the user's point of view. It is less error-prone (less code and fewer bugs) than traditional retained-mode interfaces, and lends itself to creating dynamic user interfaces. Check out the Wiki's About the IMGUI paradigm section for more details.

Dear ImGui outputs vertex buffers and command lists that you can easily render in your application. The number of draw calls and state changes required to render them is fairly small. Because Dear ImGui doesn't know or touch graphics state directly, you can call its functions anywhere in your code (e.g. in the middle of a running algorithm, or in the middle of your own rendering process).

Refer to the sample applications in the examples/ folder for instructions on how to integrate Dear ImGui with your existing codebase.

A common misunderstanding is to mistake immediate mode GUI for immediate mode rendering, which usually implies hammering your driver/GPU with a bunch of inefficient draw calls and state changes as the GUI functions are called. This is NOT what Dear ImGui does. Dear ImGui outputs vertex buffers and a small list of draw calls batches. It never touches your GPU directly. The draw call batches are decently optimal and you can render them later, in your app or even remotely.

Releases & Changelogs

See Releases page for decorated Changelogs. Reading the changelogs is a good way to keep up to date with the things Dear ImGui has to offer, and maybe will give you ideas of some features that you've been ignoring until now!

Demo

Calling the `ImGui::ShowDemoWindow()` function will create a demo window showcasing a variety of features and examples. The code is always available for reference in `imgui_demo.cpp`. Here's how the demo looks.

You should be able to build the examples from sources. If you don't, let us know! If you want to have a quick look at some Dear ImGui features, you can download Windows binaries of the demo app here:

- imgui-demo-binaries-20241211.zip (Windows, 1.91.6, built 2024/11/11, master) or older binaries.

The demo applications are not DPI aware so expect some blurriness on a 4K screen. For DPI awareness in your application, you can load/reload your font at a different scale and scale your style with `style.ScaleAllSizes()` (see FAQ).

Getting Started & Integration

See the Getting Started guide for details.

On most platforms and when using C++, **you should be able to use a combination of the imgui_impl_xxxx backends without modification** (e.g. `imgui_impl_win32.cpp` + `imgui_impl_dx11.cpp`). If your engine supports multiple platforms, consider using more imgui_impl_xxxx files instead of rewriting them: this will be less work for you, and you can get Dear ImGui running immediately. You can *later* decide to rewrite a custom backend using your custom engine functions if you wish so.

Integrating Dear ImGui within your custom engine is a matter of 1) wiring mouse/keyboard/gamepad inputs 2) uploading a texture to your GPU/render engine 3) providing a render function that can bind textures and render textured triangles, which is essentially what Backends are doing. The examples/ folder is populated with applications doing just that: setting up a window and using backends. If you follow the Getting Started guide it should in theory take you less than an hour to integrate Dear ImGui. **Make sure to spend time reading the FAQ, comments, and the examples applications!**

Officially maintained backends/bindings (in repository):

- Renderers: DirectX9, DirectX10, DirectX11, DirectX12, Metal, OpenGL/ES/ES2, SDL_GPU, SDL_Renderer2/3, Vulkan, WebGPU.
- Platforms: GLFW, SDL2/SDL3, Win32, Glut, OSX, Android.
- Frameworks: Allegro5, Emscripten.

Third-party backends/bindings wiki page:

- Languages: C, C# and: Beef, ChaiScript, CovScript, Crystal, D, Go, Haskell, Haxe/hxcpp, Java, JavaScript, Julia, Kotlin, Lobster, Lua, Nim, Odin, Pascal, PureBasic, Python, ReaScript, Ruby, Rust, Swift, Zig...
- Frameworks: AGS/Adventure Game Studio, Amethyst, Blender, bsf, Cinder, Cocos2d-x, Defold, Diligent Engine, Ebiten, Flexium, GML/Game Maker Studio, GLEQ, Godot, GTK3, Irrlicht Engine, JUCE, LÖVE+LUA, Mach Engine, Magnum, Marmalade, Monogame, NanoRT, nCine, Nim Game Lib, Nintendo 3DS/Switch/WiiU (homebrew), Ogre, openFrameworks, OSG/OpenSceneGraph, Orx, Photoshop, px_render, Qt/QtDirect3D, raylib, SFML, Sokol, Unity, Unreal Engine 4/5, UWP, vtk, VulkanHpp, VulkanSceneGraph, Win32 GDI, WxWidgets.
- Many bindings are auto-generated (by good old cimgui or newer/experimental dear_bindings), you can use their metadata output to generate bindings for other languages.

Useful Extensions/Widgets wiki page:

- Automation/testing, Text editors, node editors, timeline editors, plotting, software renderers, remote network access, memory editors, gizmos, etc. Notable and well supported extensions include ImPlot and Dear ImGui Test Engine.

Also see Wiki for more links and ideas.

Gallery

Examples projects using Dear ImGui: Tracy (profiler), ImHex (hex editor/data analysis), RemedyBG (debugger) and hundreds of others.

For more user-submitted screenshots of projects using Dear ImGui, check out the Gallery Threads!

For a list of third-party widgets and extensions, check out the Useful Extensions/Widgets wiki page.

Support, Frequently Asked Questions (FAQ)

See: Frequently Asked Questions (FAQ) where common questions are answered.

See: Getting Started and Wiki for many links, references, articles.

See: Articles about the IMGUI paradigm to read/learn about the Immediate Mode GUI paradigm.

See: Upcoming Changes.

See: Dear ImGui Test Engine + Test Suite for Automation & Testing.

For the purposes of getting search engines to crawl the wiki, here's a link to the Crawlable Wiki (not for humans, here's why).

Getting started? For first-time users having issues compiling/linking/running or issues loading fonts, please use GitHub Discussions. For ANY other questions, bug reports, requests, feedback, please post on GitHub Issues. Please read and fill the New Issue template carefully.

Private support is available for paying business customers (E-mail: *contact @ dearimgui dot com*).

Which version should I get?

We occasionally tag Releases (with nice releases notes) but it is generally safe and recommended to sync to latest `master` or `docking` branch. The library is fairly stable and regressions tend to be fixed fast when reported. Advanced users may want to use the `docking` branch with Multi-Viewport and Docking features. This branch is kept in sync with master regularly.

Who uses Dear ImGui?

See the Quotes, Funding & Sponsors, and Software using Dear ImGui Wiki pages for an idea of who is using Dear ImGui. Please add your game/software if you can! Also, see the Gallery Threads!

How to help

How can I help?

- See GitHub Forum/Issues.
- You may help with development and submit pull requests! Please understand that by submitting a PR you are also submitting a request for the maintainer to review your code and then take over its maintenance forever. PR should be crafted both in the interest of the end-users and also to ease the maintainer into understanding and accepting it.
- See Help wanted on the Wiki for some more ideas.
- Be a Funding Supporter! Have your company financially support this project via invoiced sponsors/maintenance or by buying a license for Dear ImGui Test Engine (please reach out: omar AT dearimgui DOT com).

Sponsors

Ongoing Dear ImGui development is and has been financially supported by users and private sponsors.
Please see the **detailed list of current and past Dear ImGui funding supporters and sponsors** for details.
From November 2014 to December 2019, ongoing development has also been financially supported by its users on Patreon and through individual donations.

THANK YOU to all past and present supporters for helping to keep this project alive and thriving!

Dear ImGui is using software and services provided free of charge for open source projects:

- PVS-Studio for static analysis (supports C/C++/C#/Java).
- GitHub actions for continuous integration systems.
- OpenCppCoverage for code coverage analysis.

Credits

Developed by Omar Cornut and every direct or indirect contributors to the GitHub. The early version of this library was developed with the support of Media Molecule and first used internally on the game Tearaway (PS Vita).

Recurring contributors include Rokas Kupstys @rokups (2020-2022): a good portion of work on automation system and regression tests now available in Dear ImGui Test Engine.

Maintenance/support contracts, sponsoring invoices and other B2B transactions are hosted and handled by Disco Hello.

Omar: "I first discovered the IMGUI paradigm at Q-Games where Atman Binstock had dropped his own simple implementation in the codebase, which I spent quite some time improving and thinking about. It turned out that Atman was exposed to the concept directly by working with Casey. When I moved to Media Molecule I rewrote a new library trying to overcome the flaws and limitations of the first one I've worked with. It became this library and since then I have spent an unreasonable amount of time iterating and improving it."

Embeds ProggyClean.ttf font by Tristan Grimmer (MIT license).
Embeds stb_textedit.h, stb_truetype.h, stb_rect_pack.h by Sean Barrett (public domain).

Inspiration, feedback, and testing for early versions: Casey Muratori, Atman Binstock, Mikko Mononen, Emmanuel Briney, Stefan Kamoda, Anton Mikhailov, Matt Willis. Also thank you to everyone posting feedback, questions and patches on GitHub.

License

Dear ImGui is licensed under the MIT License, see LICENSE.txt for more information.

Mostrar ventana glfw

```
#include <GLFW/glfw3.h>
#include <stdio.h>

int main(void)
{
    // Inicializar GLFW
    if (!glfwInit())
    {
        fprintf(stderr, "Fallo al inicializar GLFW\n");
        return -1;
    }

    // Crear una ventana de 800x600 píxeles
    GLFWwindow* window = glfwCreateWindow(800, 600, "Ventana con GLFW", NULL, NULL);

    if (!window)
    {
        fprintf(stderr, "Fallo al crear la ventana GLFW\n");
        glfwTerminate();
        return -1;
    }

    // Establecer el contexto actual
    glfwMakeContextCurrent(window);
```

```
    // Bucle principal: se ejecuta mientras la ventana no haya sido cerrada
    while (!glfwWindowShouldClose(window))
    {
        // Manejar eventos (como clics o teclas)
        glfwPollEvents();

        // Aquí iría el código para renderizar (OpenGL)

        // Intercambiar los buffers (doble buffering)
        glfwSwapBuffers(window);
    }

    // Limpiar recursos
    glfwDestroyWindow(window);
    glfwTerminate();

    return 0;
}
```

Compilar:

sudo apt-get install libglfw3-dev libgl1 mesa-glx

g++ main.c -o ventana_glfw -I/usr/local/include -L/usr/local/lib -lglfw -framework OpenGL

Ventana con controles

```
#include <imgui.h>
#include <imgui_impl_glfw.h>
#include <imgui_impl_opengl3.h>
#include <GLFW/glfw3.h>
#include <stdio.h>

int main(void)
{
    // Inicializar GLFW
    if (!glfwInit())
    {
        fprintf(stderr, "Fallo al inicializar GLFW\n");
        return -1;
    }

    // Crear ventana GLFW
    GLFWwindow* window = glfwCreateWindow(800, 600, "Ventana con ImGui (sin GLAD)",
NULL, NULL);
    if (!window)
    {
        fprintf(stderr, "Fallo al crear la ventana GLFW\n");
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSwapInterval(1); // Habilita VSync

    // Inicializar ImGui
    IMGUI_CHECKVERSION();
    ImGui::CreateContext();
    ImGuiIO& io = ImGui::GetIO(); (void)io;
```

```
ImGui::StyleColorsDark();

// Backend de ImGui para GLFW + OpenGL
ImGui_ImplGlfw_InitForOpenGL(window, true);
ImGui_ImplOpenGL3_Init("#version 120"); // Usa GLSL 1.20 (compatible con OpenGL 2.1)

bool show_demo_window = false;
bool my_bool = false;
float my_float = 0.0f;

// Bucle principal
while (!glfwWindowShouldClose(window))
{
    glfwPollEvents();

    // Iniciar frame de ImGui
    ImGui_ImplOpenGL3_NewFrame();
    ImGui_ImplGlfw_NewFrame();
    ImGui::NewFrame();

    // Ventana personalizada
    ImGui::Begin("Mi Ventana", NULL, ImGuiWindowFlags_MenuBar);
    if (ImGui::Button("Aumentar"))
        my_float += 1.0f;

    ImGui::Checkbox("Activo", &my_bool);
    ImGui::SliderFloat("Valor", &my_float, 0.0f, 5.0f);

    ImGui::Text("FPS: %.1f", io.Framerate);
    ImGui::End();
```

```
        // Mostrar ventana de demostración opcional
        if (show_demo_window)
            ImGui::ShowDemoWindow(&show_demo_window);

        // Dibujar
        ImGui::Render();
        glClearColor(0.45f, 0.55f, 0.60f, 1.00f);
        glClear(GL_COLOR_BUFFER_BIT);
        ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());

        glfwSwapBuffers(window);
    }

    // Limpiar recursos
    ImGui_ImplOpenGL3_Shutdown();
    ImGui_ImplGlfw_Shutdown();
    ImGui::DestroyContext();

    glfwDestroyWindow(window);
    glfwTerminate();

    return 0;

}
```

Compilar:

```
g++ ./main.c -lglfw -l imgui -I/usr/include/imgui
-I/usr/include/imgui/backends/ -lGL -lstb
```

Ventana imgui en blanco.

```
#include <imgui.h>
#include <imgui_impl_glfw.h>
#include <imgui_impl_opengl3.h>

#include <GLFW/glfw3.h>
#include <stdio.h>

int main(void)
{
    // Inicializar GLFW
    if (!glfwInit())
    {
        fprintf(stderr, "Fallo al inicializar GLFW\n");
        return -1;
    }

    // Crear ventana GLFW
    GLFWwindow* window = glfwCreateWindow(800, 600, "Ventana Limpia con ImGui", NULL,
NULL);
    if (!window)
    {
        fprintf(stderr, "Fallo al crear la ventana GLFW\n");
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSwapInterval(1); // Habilita VSync

    // Inicializar ImGui
```

```
IMGUI_CHECKVERSION();
ImGui::CreateContext();
ImGuiIO& io = ImGui::GetIO(); (void)io;

ImGui::StyleColorsDark(); // Puedes usar StyleColorsLight() si prefieres

// Backend de ImGui para GLFW + OpenGL
ImGui_ImplGlfw_InitForOpenGL(window, true);
ImGui_ImplOpenGL3_Init("#version 120"); // GLSL versión 1.20 compatible con OpenGL 2.1

// Bucle principal
while (!glfwWindowShouldClose(window))
{
    glfwPollEvents();

    // Nuevo frame de ImGui
    ImGui_ImplOpenGL3_NewFrame();
    ImGui_ImplGlfw_NewFrame();
    ImGui::NewFrame();

    // Aquí no añadimos ninguna interfaz, dejamos la pantalla limpia

    // Renderizado final
    ImGui::Render();
    glClearColor(0.45f, 0.55f, 0.60f, 1.00f); // Color de fondo
    glClear(GL_COLOR_BUFFER_BIT);
    ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());

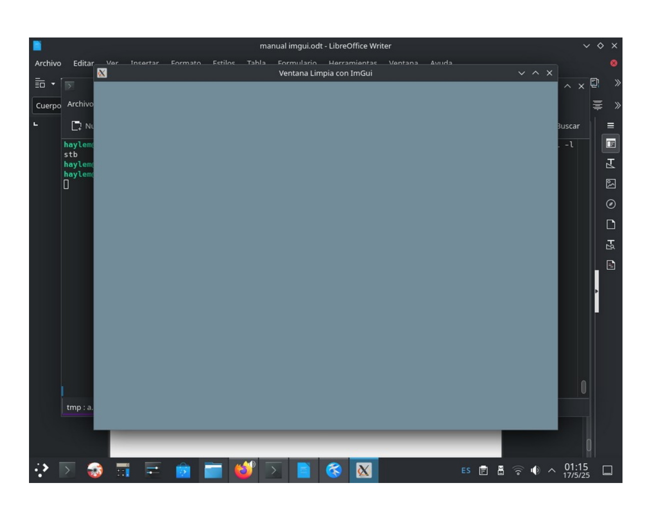
    glfwSwapBuffers(window);
}

// Limpiar recursos
```

```
    ImGui_ImplOpenGL3_Shutdown();
    ImGui_ImplGlfw_Shutdown();
    ImGui::DestroyContext();

    glfwDestroyWindow(window);
    glfwTerminate();

    return 0;
}
```



Ventana con botón saludar que muestra saludo en terminal.

```
#include <imgui.h>
#include <imgui_impl_glfw.h>
#include <imgui_impl_opengl3.h>

#include <GLFW/glfw3.h>
#include <stdio.h>

int main(void)
{
    // Inicializar GLFW
    if (!glfwInit())
```

```
    {
        fprintf(stderr, "Fallo al inicializar GLFW\n");
        return -1;
    }

    // Crear ventana GLFW
    GLFWwindow* window = glfwCreateWindow(800, 600, "Ventana Limpia con ImGui", NULL,
NULL);
    if (!window)
    {
        fprintf(stderr, "Fallo al crear la ventana GLFW\n");
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSwapInterval(1); // Habilita VSync

    // Inicializar ImGui
    IMGUI_CHECKVERSION();
    ImGui::CreateContext();
    ImGuiIO& io = ImGui::GetIO(); (void)io;

    ImGui::StyleColorsDark(); // Puedes usar StyleColorsLight() si prefieres

    // Backend de ImGui para GLFW + OpenGL
    ImGui_ImplGlfw_InitForOpenGL(window, true);
    ImGui_ImplOpenGL3_Init("#version 120"); // GLSL versión 1.20 compatible con OpenGL 2.1

    // Bucle principal
    while (!glfwWindowShouldClose(window))
    {
        glfwPollEvents();
```

```
    // Nuevo frame de ImGui
    ImGui_ImplOpenGL3_NewFrame();
    ImGui_ImplGlfw_NewFrame();
    ImGui::NewFrame();

    // Ventana con botón
    ImGui::Begin("Acciones");
    if (ImGui::Button("Saludar"))
    {
        printf("¡Hola desde ImGui!\n");
    }
    ImGui::End();

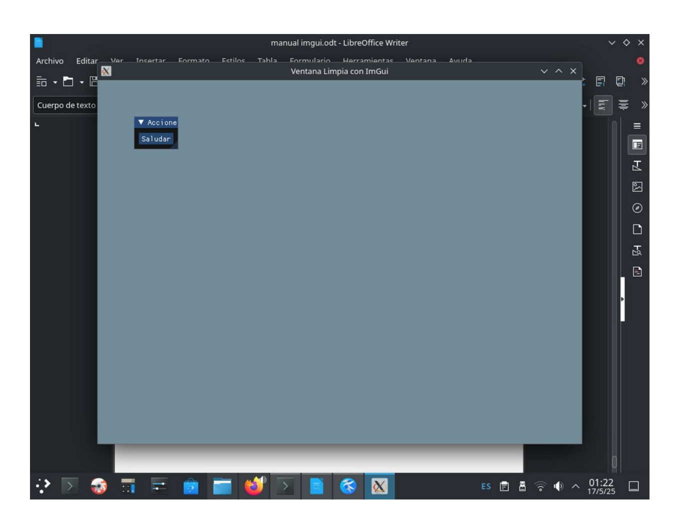
    // Renderizado final
    ImGui::Render();
    glClearColor(0.45f, 0.55f, 0.60f, 1.00f); // Color de fondo
    glClear(GL_COLOR_BUFFER_BIT);
    ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());

    glfwSwapBuffers(window);
}

// Limpiar recursos
ImGui_ImplOpenGL3_Shutdown();
ImGui_ImplGlfw_Shutdown();
ImGui::DestroyContext();
```

```
    glfwDestroyWindow(window);
    glfwTerminate();

    return 0;
}
```



Botones de colores

```
#include <imgui.h>
#include <imgui_impl_glfw.h>
#include <imgui_impl_opengl3.h>

#include <GLFW/glfw3.h>
#include <stdio.h>

int main(void)
{
    // Inicializar GLFW
    if (!glfwInit())
    {
        fprintf(stderr, "Fallo al inicializar GLFW\n");
        return -1;
```

```
    }

    // Crear ventana GLFW
    GLFWwindow* window = glfwCreateWindow(800, 600, "Ventana con Botones de Colores",
NULL, NULL);
    if (!window)
    {
        fprintf(stderr, "Fallo al crear la ventana GLFW\n");
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSwapInterval(1); // Habilita VSync

    // Inicializar ImGui
    IMGUI_CHECKVERSION();
    ImGui::CreateContext();
    ImGuiIO& io = ImGui::GetIO(); (void)io;

    ImGui::StyleColorsDark();

    // Backend de ImGui para GLFW + OpenGL
    ImGui_ImplGlfw_InitForOpenGL(window, true);
    ImGui_ImplOpenGL3_Init("#version 120");

    // Bucle principal
    while (!glfwWindowShouldClose(window))
    {
        glfwPollEvents();

        // Nuevo frame de ImGui
        ImGui_ImplOpenGL3_NewFrame();
```

```
ImGui_ImplGlfw_NewFrame();
ImGui::NewFrame();

// Ventana con botones de colores
ImGui::Begin("Botones de Colores");

// Botón rojo
ImGui::PushStyleColor(ImGuiCol_Button, ImVec4(1.0f, 0.0f, 0.0f, 1.0f));
ImGui::Button("Rojo", ImVec2(150, 50));
ImGui::PopStyleColor();

ImGui::Spacing();

// Botón verde
ImGui::PushStyleColor(ImGuiCol_Button, ImVec4(0.0f, 1.0f, 0.0f, 1.0f));
ImGui::Button("Verde", ImVec2(150, 50));
ImGui::PopStyleColor();

ImGui::Spacing();

// Botón azul
ImGui::PushStyleColor(ImGuiCol_Button, ImVec4(0.0f, 0.0f, 1.0f, 1.0f));
ImGui::Button("Azul", ImVec2(150, 50));
ImGui::PopStyleColor();

ImGui::Spacing();

// Botón amarillo
ImGui::PushStyleColor(ImGuiCol_Button, ImVec4(1.0f, 1.0f, 0.0f, 1.0f));
ImGui::Button("Amarillo", ImVec2(150, 50));
ImGui::PopStyleColor();
```

```
        ImGui::End();

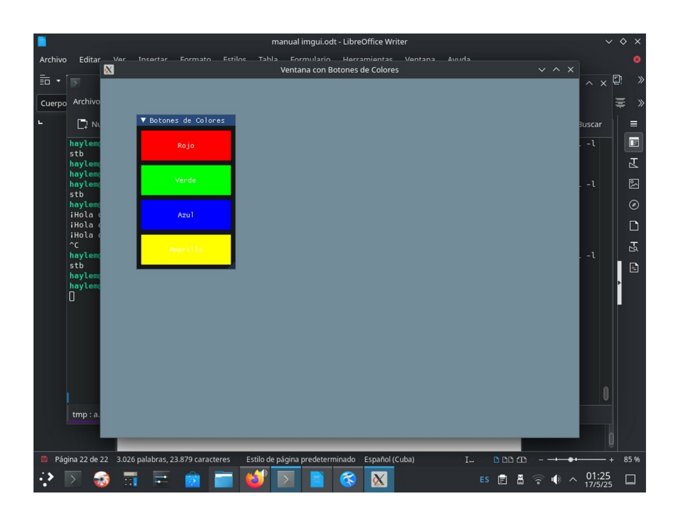
        // Renderizado final
        ImGui::Render();
        glClearColor(0.45f, 0.55f, 0.60f, 1.00f); // Color de fondo
        glClear(GL_COLOR_BUFFER_BIT);
        ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());

        glfwSwapBuffers(window);
    }

    // Limpiar recursos
    ImGui_ImplOpenGL3_Shutdown();
    ImGui_ImplGlfw_Shutdown();
    ImGui::DestroyContext();

    glfwDestroyWindow(window);
    glfwTerminate();

    return 0;
}
```



Los controles

```
#include "imgui.h"
#include "imgui_impl_glfw.h"
#include "imgui_impl_opengl3.h"

#include <GLFW/glfw3.h>
#include <stdio.h>

int main(void)
{
    // Inicializar GLFW
    if (!glfwInit())
    {
        fprintf(stderr, "Fallo al inicializar GLFW\n");
        return -1;
    }

    // Crear ventana GLFW
    GLFWwindow* window = glfwCreateWindow(1000, 800, "ImGui - Todos los Controles",
NULL, NULL);
    if (!window)
    {
        fprintf(stderr, "Fallo al crear la ventana GLFW\n");
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSwapInterval(1); // Habilita VSync

    // Inicializar ImGui
```

```
IMGUI_CHECKVERSION();
ImGui::CreateContext();
ImGuiIO& io = ImGui::GetIO(); (void)io;

ImGui::StyleColorsDark(); // Puedes usar StyleColorsLight() si prefieres

// Backend de ImGui para GLFW + OpenGL
ImGui_ImplGlfw_InitForOpenGL(window, true);
ImGui_ImplOpenGL3_Init("#version 120");

// Variables de los controles
bool checkboxValue = false;
int radioButtonValue = 0;
int sliderInt = 50;
float sliderFloat = 0.5f;
char inputText[128] = "Escribe aquí";
const char* items[] = { "Opción 1", "Opción 2", "Opción 3" };
int itemSelected = 0;
ImVec4 colorPickerValue = ImVec4(0.4f, 0.7f, 0.9f, 1.0f);

// Bucle principal
while (!glfwWindowShouldClose(window))
{
    glfwPollEvents();

    // Nuevo frame de ImGui
    ImGui_ImplOpenGL3_NewFrame();
    ImGui_ImplGlfw_NewFrame();
    ImGui::NewFrame();

    // Ventana principal con todos los controles
    ImGui::Begin("Todos los Controles de ImGui");
```

```
// Botones
ImGui::Text("Botones:");
if (ImGui::Button("Botón Simple")) {
    printf("Botón simple presionado\n");
}
ImGui::SameLine();
if (ImGui::Button("Botón con SameLine")) {
    printf("Botón en la misma línea\n");
}
ImGui::Separator();

// Checkbox
ImGui::Text("Checkbox:");
ImGui::Checkbox("Activo/Inactivo", &checkboxValue);
ImGui::Separator();

// Radio Buttons
ImGui::Text("Radio Buttons:");
ImGui::RadioButton("Opción A", &radioButtonValue, 0);
ImGui::SameLine();
ImGui::RadioButton("Opción B", &radioButtonValue, 1);
ImGui::SameLine();
ImGui::RadioButton("Opción C", &radioButtonValue, 2);
ImGui::Separator();

// Sliders enteros y flotantes
ImGui::Text("Sliders:");
ImGui::SliderInt("Entero (0-100)", &sliderInt, 0, 100);
ImGui::SliderFloat("Flotante (0.0-1.0)", &sliderFloat, 0.0f, 1.0f);
ImGui::Separator();
```

```
// Input Text
ImGui::Text("Input Text:");
ImGui::InputText("Texto", inputText, IM_ARRAYSIZE(inputText));
ImGui::Separator();

// ComboBox / Dropdown
ImGui::Text("ComboBox:");
ImGui::Combo("Selecciona opción", &itemSelected, items, IM_ARRAYSIZE(items));
ImGui::Separator();

// Color Picker
ImGui::Text("Color Picker:");
ImGui::ColorEdit4("Elige color", (float*)&colorPickerValue);
ImGui::Separator();

// Espaciado y botón final
ImGui::Spacing();
if (ImGui::Button("Imprimir Valores Consola")) {
    printf("Checkbox: %s\n", checkboxValue ? "true" : "false");
    printf("Radio Button: %d\n", radioButtonValue);
    printf("Slider Entero: %d\n", sliderInt);
    printf("Slider Flotante: %.2f\n", sliderFloat);
    printf("Texto ingresado: %s\n", inputText);
    printf("Item seleccionado: %s\n", items[itemSelected]);
    printf("Color seleccionado: (%.2f, %.2f, %.2f, %.2f)\n",
           colorPickerValue.x, colorPickerValue.y,
           colorPickerValue.z, colorPickerValue.w);
}

ImGui::End();
```

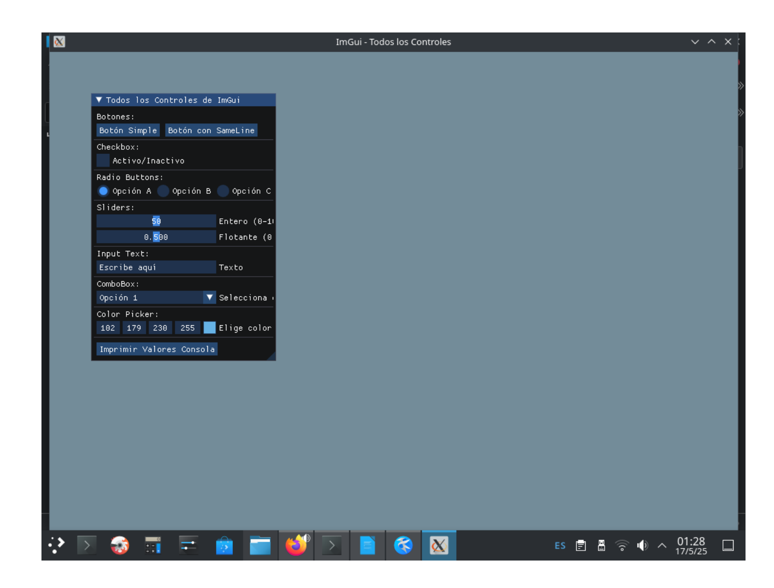
```
    // Renderizado final
        ImGui::Render();
        glClearColor(0.45f, 0.55f, 0.60f, 1.00f); // Color de fondo
        glClear(GL_COLOR_BUFFER_BIT);
        ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());

        glfwSwapBuffers(window);
    }

    // Limpiar recursos
    ImGui_ImplOpenGL3_Shutdown();
    ImGui_ImplGlfw_Shutdown();
    ImGui::DestroyContext();

    glfwDestroyWindow(window);
    glfwTerminate();

    return 0;
}
```



Campos entrada. Sumar 2 enteros

```
#include "imgui.h"
#include "imgui_impl_glfw.h"
#include "imgui_impl_opengl3.h"

#include <GLFW/glfw3.h>
#include <stdio.h>

int main(void)
{
    // Inicializar GLFW
    if (!glfwInit())
    {
        fprintf(stderr, "Fallo al inicializar GLFW\n");
        return -1;
    }

    // Crear ventana GLFW
    GLFWwindow* window = glfwCreateWindow(400, 300, "Ejemplo Suma con ImGui", NULL, 
NULL);
    if (!window)
    {
        fprintf(stderr, "Fallo al crear la ventana GLFW\n");
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSwapInterval(1); // Habilita VSync

    // Inicializar ImGui
    IMGUI_CHECKVERSION();
```

```
ImGui::CreateContext();
ImGuiIO& io = ImGui::GetIO(); (void)io;

ImGui::StyleColorsDark(); // Puedes usar StyleColorsLight() si prefieres

// Backend de ImGui para GLFW + OpenGL
ImGui_ImplGlfw_InitForOpenGL(window, true);
ImGui_ImplOpenGL3_Init("#version 120");

// Variables para los números y el resultado
int numero1 = 0;
int numero2 = 0;
int resultado = 0;
bool mostrarResultado = false;

// Bucle principal
while (!glfwWindowShouldClose(window))
{
    glfwPollEvents();

    // Nuevo frame de ImGui
    ImGui_ImplOpenGL3_NewFrame();
    ImGui_ImplGlfw_NewFrame();
    ImGui::NewFrame();

    // Ventana con inputs y botón
    ImGui::Begin("Calculadora Simple");

    ImGui::Text("Ingrese dos números:");

    ImGui::InputInt("Número 1", &numero1);
    ImGui::InputInt("Número 2", &numero2);
```

```
    if (ImGui::Button("Sumar"))
    {
        resultado = numero1 + numero2;
        mostrarResultado = true;
    }

    if (mostrarResultado)
    {
        ImGui::SameLine();
        ImGui::TextColored(ImVec4(0, 1, 0, 1), "Resultado: %d", resultado);
    }

    ImGui::End();

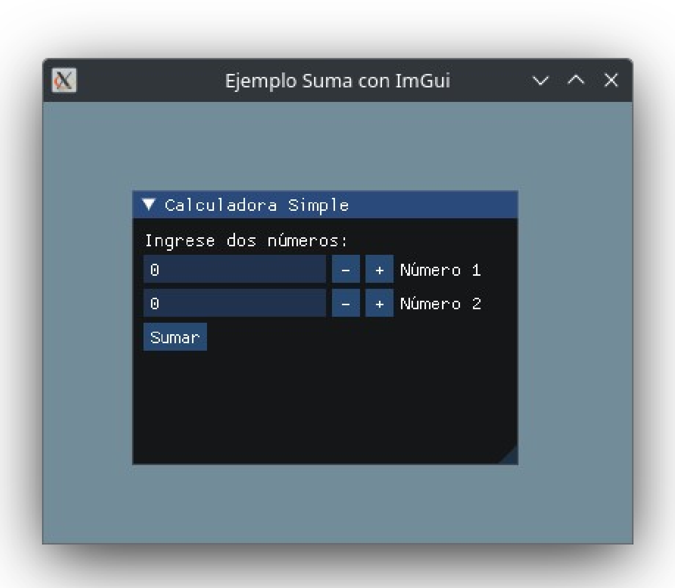
    // Renderizado final
    ImGui::Render();
    glClearColor(0.45f, 0.55f, 0.60f, 1.00f); // Color de fondo
    glClear(GL_COLOR_BUFFER_BIT);
    ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());

    glfwSwapBuffers(window);
}

// Limpiar recursos
ImGui_ImplOpenGL3_Shutdown();
ImGui_ImplGlfw_Shutdown();
ImGui::DestroyContext();
```

```
    glfwDestroyWindow(window);
    glfwTerminate();

    return 0;
}
```



Suma flotantes

```
#include "imgui.h"
#include "imgui_impl_glfw.h"
#include "imgui_impl_opengl3.h"

#include <GLFW/glfw3.h>
#include <stdio.h>

int main(void)
{
    // Inicializar GLFW
    if (!glfwInit())
```

```
    {
        fprintf(stderr, "Fallo al inicializar GLFW\n");
        return -1;
    }

    // Crear ventana GLFW
    GLFWwindow* window = glfwCreateWindow(400, 300, "Ejemplo Suma con Floats", NULL,
NULL);
    if (!window)
    {
        fprintf(stderr, "Fallo al crear la ventana GLFW\n");
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSwapInterval(1); // Habilita VSync

    // Inicializar ImGui
    IMGUI_CHECKVERSION();
    ImGui::CreateContext();
    ImGuiIO& io = ImGui::GetIO(); (void)io;

    ImGui::StyleColorsDark(); // Puedes usar StyleColorsLight() si prefieres

    // Backend de ImGui para GLFW + OpenGL
    ImGui_ImplGlfw_InitForOpenGL(window, true);
    ImGui_ImplOpenGL3_Init("#version 120");

    // Variables para los números y el resultado
    float numero1 = 0.0f;
    float numero2 = 0.0f;
    float resultado = 0.0f;
```

```
bool mostrarResultado = false;

// Bucle principal
while (!glfwWindowShouldClose(window))
{
    glfwPollEvents();

    // Nuevo frame de ImGui
    ImGui_ImplOpenGL3_NewFrame();
    ImGui_ImplGlfw_NewFrame();
    ImGui::NewFrame();

    // Ventana con inputs y botón
    ImGui::Begin("Calculadora con Flotantes");

    ImGui::Text("Ingrese dos números decimales:");

    ImGui::InputFloat("Número 1", &numero1, 0.01f, 0.1f, "%.2f"); // Formato con 2 decimales
    ImGui::InputFloat("Número 2", &numero2, 0.01f, 0.1f, "%.2f");

    if (ImGui::Button("Sumar"))
    {
        resultado = numero1 + numero2;
        mostrarResultado = true;
    }

    if (mostrarResultado)
    {
        ImGui::SameLine();
        ImGui::TextColored(ImVec4(0, 1, 0, 1), "Resultado: %.2f", resultado);
    }
```

```
        ImGui::End();

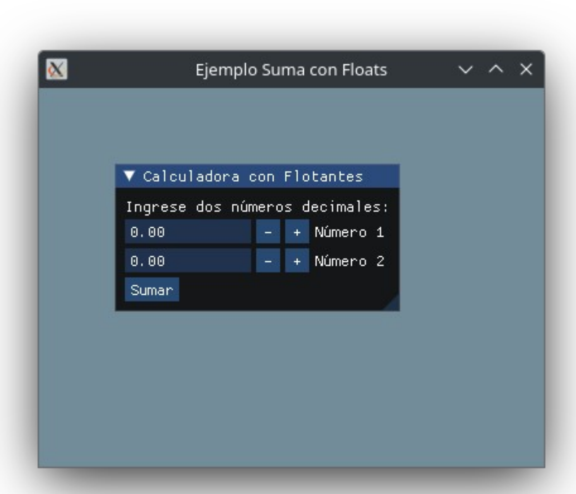
        // Renderizado final
        ImGui::Render();
        glClearColor(0.45f, 0.55f, 0.60f, 1.00f); // Color de fondo
        glClear(GL_COLOR_BUFFER_BIT);
        ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());

        glfwSwapBuffers(window);
    }

    // Limpiar recursos
    ImGui_ImplOpenGL3_Shutdown();
    ImGui_ImplGlfw_Shutdown();
    ImGui::DestroyContext();

    glfwDestroyWindow(window);
    glfwTerminate();

    return 0;
}
```



Tipos de Contenedores en ImGui

ImGui no usa un sistema de layouts como Qt o HTML/CSS, sino que se basa en un **sistema de layout imperativo y secuencial** . Es decir: **lo que dibujas primero aparece primero** . Pero ofrece herramientas muy útiles para estructurar tu interfaz:

1. `ImGui::Begin() / ImGui::End()`

Es el contenedor básico. Crea una ventana flotante con título, botones de cierre, redimensionamiento opcional, etc.

```
ImGui::Begin("Mi Ventana");
// Aquí van tus widgets
ImGui::End();
```

2. `ImGui::BeginChild() / ImGui::EndChild()`

Crea un área scrollable dentro de otra ventana (útil para listas largas o grupos de widgets).

```
ImGui::BeginChild("AreaScroll", ImVec2(0, 100), true);
ImGui::Text("Contenido scrollable...");
ImGui::EndChild();
```

`ImVec2(0, 100)` → anchura automática (`0`) y altura fija.

- `true` → borde visible.

3. `ImGui::Columns(n)`

Permite dividir el espacio actual en varias columnas (muy útil para formularios o tablas simples).

```
ImGui::Columns(2); // Dos columnas
ImGui::Text("Nombre:");
ImGui::NextColumn();
ImGui::InputText("##nombre", nombreBuffer, 128);
ImGui::NextColumn();
ImGui::Columns(1); // Volver a una columna
```

4. `ImGui::TabBar("tabs") / ImGui::TabItem()`

Para crear pestañas.

```
if (ImGui::BeginTabBar("Opciones"))
{
    if (ImGui::BeginTabItem("Pestaña 1"))
    {
        ImGui::Text("Contenido de la pestaña 1");
        ImGui::EndTabItem();
    }

    if (ImGui::BeginTabItem("Pestaña 2"))
    {
        ImGui::Text("Contenido de la pestaña 2");
        ImGui::EndTabItem();
    }

    ImGui::EndTabBar();
}
```

5. `ImGui::TreeNode() / ImGui::TreePop()`

Para crear menús desplegables o estructuras jerárquicas (ideales para configuraciones o árboles).

```
if (ImGui::TreeNode("Configuración Avanzada"))
{
    ImGui::Checkbox("Opción A", &opcionA);
    ImGui::Checkbox("Opción B", &opcionB);
    ImGui::SliderFloat("Brillo", &brillo, 0.0f, 1.0f);
    ImGui::TreeNodePop();
}
```

Herramientas de Organización

1. **`ImGui::Spacing()`**

Inserta un espacio vertical entre elementos.

```
ImGui::Button("Botón 1");
ImGui::Spacing();
ImGui::Button("Botón 2");
```

2. `ImGui::SameLine()`

Coloca el siguiente widget en la misma línea que el anterior.

```
ImGui::Button("Guardar");
ImGui::SameLine();
ImGui::Button("Cancelar");
```

3. `ImGui::Separator()`

Dibuja una línea horizontal para separar secciones.

```
ImGui::Text("Sección 1");
ImGui::Separator();
ImGui::Text("Sección 2");
```

4. `ImGui::Dummy(size)`

Espaciado personalizado. Útil para alinear o dejar huecos exactos.

```
ImGui::Dummy(ImVec2(0.0f, 20.0f)); // 20px de espacio vertical
```

5. `ImGui::PushItemWidth(width)`

Controla el ancho de los siguientes widgets.

```
ImGui::PushItemWidth(150.0f);

ImGui::InputText("Texto corto", buffer, 128);

ImGui::PopItemWidth(); // Restaurar ancho por defecto
```

Para generar una ventana que **muestre todos los métodos de organización de la interfaz de ImGui** , vamos a construir un ejemplo que muestre cómo se usan las funciones principales para organizar elementos en una interfaz: `ImGui::Begin()`, `ImGui::End()`, `ImGui::Columns()`, `ImGui::SameLine()`, `ImGui::Separator()`, `ImGui::Spacing()`, `ImGui::Dummy()`, `ImGui::Indent()`, `ImGui::Unindent()`, `ImGui::Group()`, `ImGui::BeginChild()` / `ImGui::EndChild()`, y más.

```
#include <imgui.h>

#include <imgui_impl_glfw.h>

#include <imgui_impl_opengl3.h>


#include <GLFW/glfw3.h>

#include <stdio.h>


int main(void)

{

    // Inicializar GLFW

    if (!glfwInit())

    {

        fprintf(stderr, "Fallo al inicializar GLFW\n");

        return -1;

    }


    // Crear ventana GLFW

    GLFWwindow* window = glfwCreateWindow(1000, 800, "Ejemplo de Organización de Interfaz
con ImGui", NULL, NULL);
```

```
if (!window)
{
    fprintf(stderr, "Fallo al crear la ventana GLFW\n");
    glfwTerminate();
    return -1;
}
glfwMakeContextCurrent(window);
glfwSwapInterval(1); // Habilita VSync

// Inicializar ImGui
IMGUI_CHECKVERSION();
ImGui::CreateContext();
ImGuiIO& io = ImGui::GetIO(); (void)io;

ImGui::StyleColorsDark();

// Backend de ImGui para GLFW + OpenGL
ImGui_ImplGlfw_InitForOpenGL(window, true);
ImGui_ImplOpenGL3_Init("#version 120");

// Bucle principal
while (!glfwWindowShouldClose(window))
{
    glfwPollEvents();

    // Nuevo frame de ImGui
    ImGui_ImplOpenGL3_NewFrame();
    ImGui_ImplGlfw_NewFrame();
    ImGui::NewFrame();

    // Ventana principal de ejemplo
    ImGui::Begin("Organización de Interfaz");
```

```
ImGui::Text("1. Uso de SameLine()");
ImGui::Button("Izquierda", ImVec2(100, 0));
ImGui::SameLine();
ImGui::Button("Derecha", ImVec2(100, 0));

ImGui::Separator();
ImGui::Text("2. Uso de Columns()");
ImGui::Columns(2, "columnas");
ImGui::Text("Columna 1");
ImGui::NextColumn();
ImGui::Text("Columna 2");
ImGui::Columns(1);

ImGui::Separator();
ImGui::Text("3. Uso de Spacing()");
ImGui::Button("Arriba", ImVec2(100, 0));
ImGui::Spacing();
ImGui::Button("Abajo", ImVec2(100, 0));

ImGui::Separator();
ImGui::Text("4. Uso de Dummy()");
ImGui::Button("Botón A", ImVec2(100, 0));
ImGui::Dummy(ImVec2(0, 20)); // Espacio vertical
ImGui::Button("Botón B", ImVec2(100, 0));

ImGui::Separator();
ImGui::Text("5. Uso de Indent/Unindent()");
ImGui::Indent();
ImGui::Button("Indentado", ImVec2(100, 0));
ImGui::Unindent();
ImGui::Button("Normal", ImVec2(100, 0));
```

```
ImGui::Separator();
ImGui::Text("6. Uso de Group()");
ImGui::BeginGroup();
ImGui::Button("Grupo 1", ImVec2(100, 0));
ImGui::Button("Grupo 2", ImVec2(100, 0));
ImGui::EndGroup();

ImGui::SameLine();
ImGui::BeginGroup();
ImGui::Button("Grupo A", ImVec2(100, 0));
ImGui::Button("Grupo B", ImVec2(100, 0));
ImGui::EndGroup();

ImGui::Separator();
ImGui::Text("7. Uso de Child Window (Ventana Interna)");
ImGui::BeginChild("Ventana Interna", ImVec2(0, 100), true);
ImGui::Text("Este es un child window.");
ImGui::EndChild();

ImGui::Separator();
ImGui::Text("8. Ejemplo combinado");
ImGui::BeginGroup();
ImGui::Button("A", ImVec2(50, 30));
ImGui::SameLine();
ImGui::Button("B", ImVec2(50, 30));
ImGui::Dummy(ImVec2(0, 10));
ImGui::Button("C", ImVec2(100, 30));
ImGui::EndGroup();

ImGui::End(); // Fin de la ventana principal
```

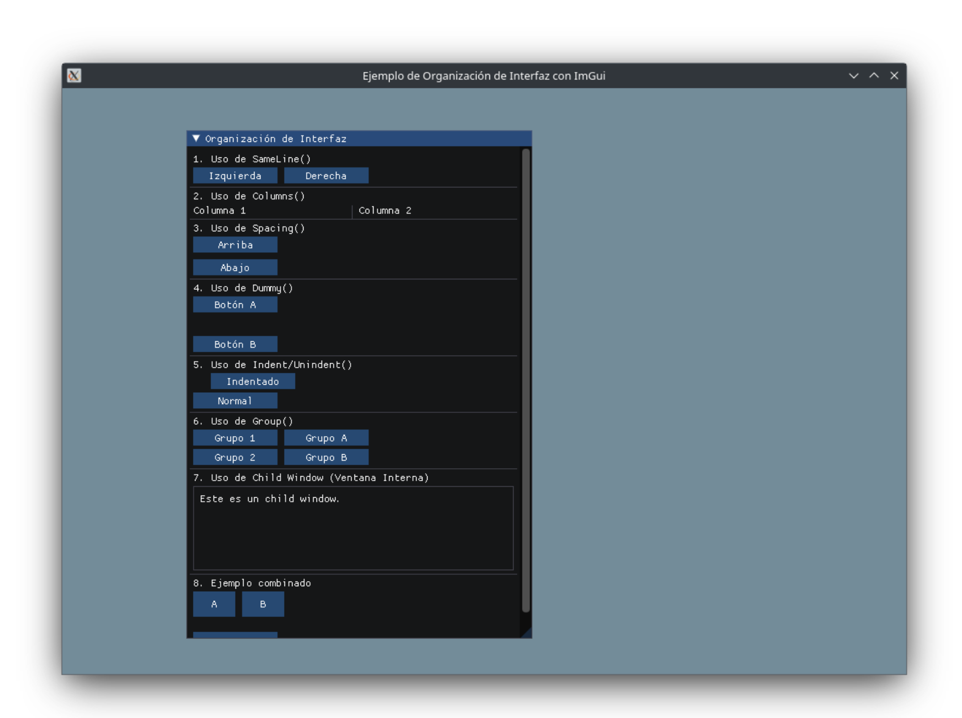
```
        // Renderizado final
        ImGui::Render();
        glClearColor(0.45f, 0.55f, 0.60f, 1.00f);
        glClear(GL_COLOR_BUFFER_BIT);
        ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());

        glfwSwapBuffers(window);
    }

    // Limpiar recursos
    ImGui_ImplOpenGL3_Shutdown();
    ImGui_ImplGlfw_Shutdown();
    ImGui::DestroyContext();

    glfwDestroyWindow(window);
    glfwTerminate();

    return 0;
}
```



Explicación breve de cada método:

`ImGui::SameLine()`	Muestra el siguiente widget en la misma línea horizontal.
`ImGui::Columns(n)`	Divide la interfaz en columnas (útil para formularios o tablas simples).
`ImGui::Separator()`	Dibuja una línea divisoria visual.
`ImGui::Spacing()`	Añade espacio proporcional entre widgets.
`ImGui::Dummy(size)`	Añade espacio fijo (útil para saltos verticales/horizontales personalizados).
`ImGui::Indent()` / `Unindent()`	Cambia la sangría del contenido.
`ImGui::Group()`	Agrupa varios widgets como una unidad lógica (para alinear o posicionar).
`ImGui::BeginChild()`	Crea una subventana dentro de otra ventana.